Warm Starting Series of Mixed-Integer Linear Programs

Sean Kelley

22 September 2022

Overview

Background

Motivation

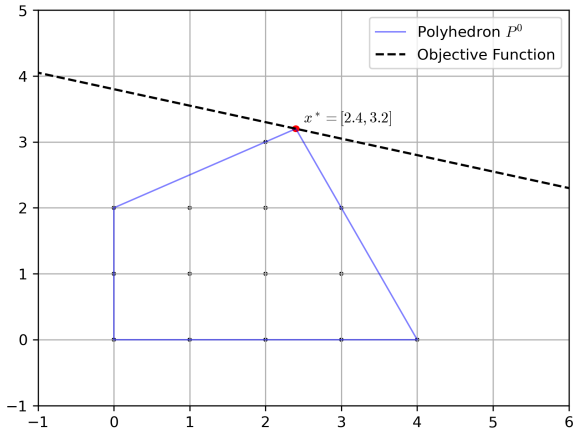
- Mixed-Integer Programming has created great value in industry.
- One important source comes from problems solved by a series of Mixed-Integer Linear Programs (MILPs).
  - Electric Grid Production Planning (Stochastic Dual Decomposition)
  - Vehicle Routing (Branch and Price)
- Many MILP instances in such series differ only by objective coefficients or bounds on their constraints.
- Solvers can leverage what they discover solving one MILP to more quickly solve a similar MILP (a.k.a. "Warm Start").
- Warm starting would enable greater performance and expand the space of tractable problems for those solved as a series of MILPs.

This presentation outlines a potential opportunity to warm start series of MILPs and discusses details towards its implementation.

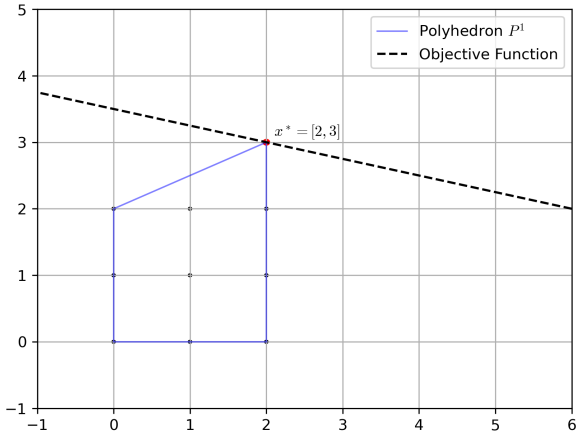
Simple Example

$$\begin{aligned} \max \quad & x + 4y \\ \text{s.t.} \quad & -\frac{x}{2} + y \leq 2, \\ & x + \frac{y}{2} \leq 4, \\ & (x, y) \in \mathbb{Z}^2_+ \end{aligned} \tag{1}$$

- Solving (1) yields Figure 1's Branch and Bound tree.
- Constraints like "$x \leq 2$ or $x \geq 3$" cause branching.
- Such constraints are called **disjunctions**.
- We refer to their collection as *the* disjunction.



Branch on $x \leq 2$ or $x \geq 3$



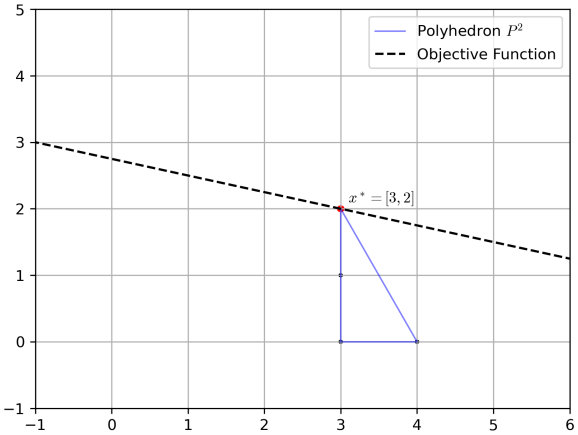


Figure 1: (1)'s Branch and Bound tree

Simple Example

$$\begin{aligned} \max \quad & x + 4y \\ \text{s.t.} \quad & -\frac{x}{2} + y \leq 3, \\ & x + \frac{y}{2} \leq 5, \\ & (x, y) \in \mathbb{Z}^2_+ \end{aligned} \tag{2}$$

- (2) loosens the RHS from (1).
- Applying (1)'s disjunction yields a partial Branch and Bound tree for (2).
- This partial tree immediately yields the solution for (2).

Branch on $x \leq 2$ or $x \geq 3$

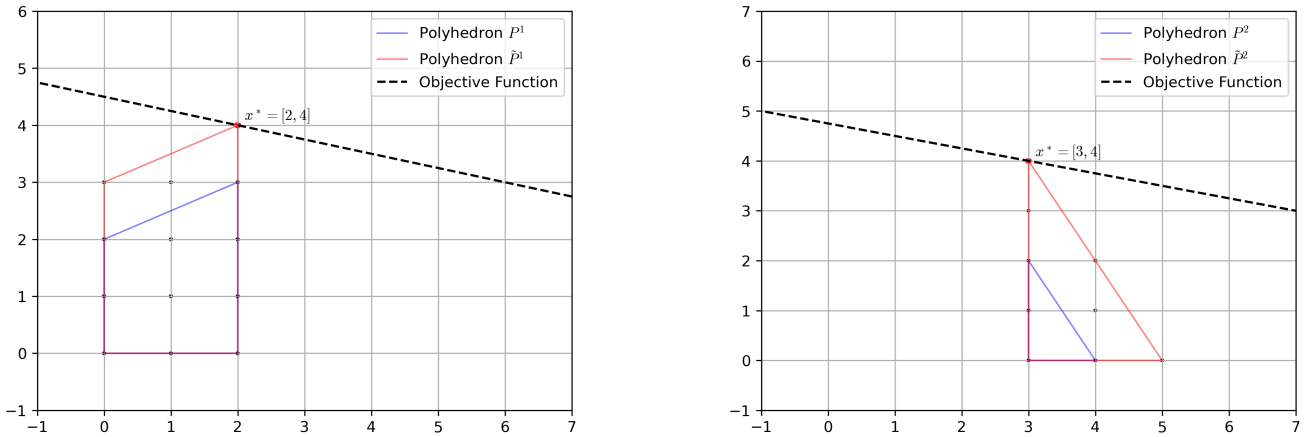
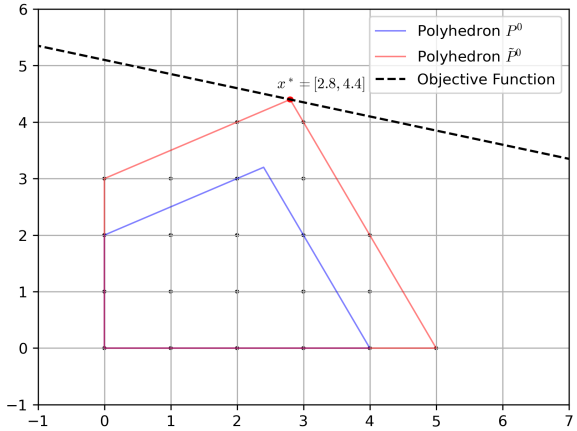


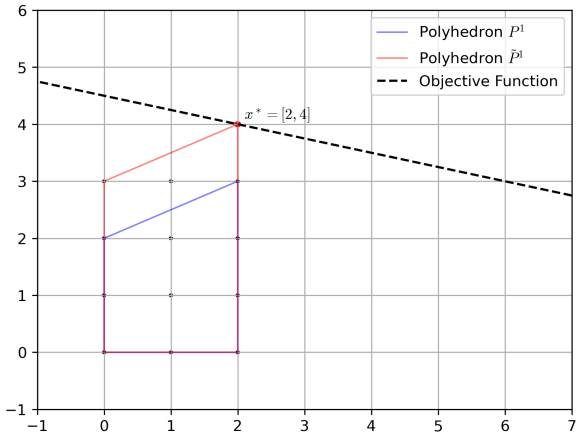
Figure 2: (2)'s partial Branch and Bound tree

Simple Example

- Solving (2) yields Figure 3's Branch and Bound tree.
- Figure 3 contains the partial tree from Figure 2.
- Applying Branch and Bound or the disjunction from (1) both yield the same solution.



Branch on $x \leq 2$ or $x \geq 3$



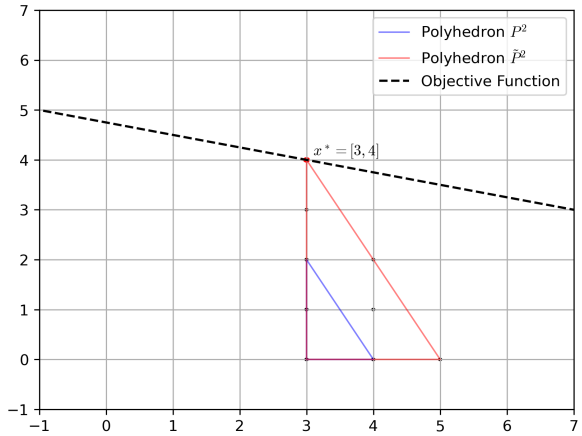


Figure 3: (2)'s full Branch and Bound tree

Applying the disjunction from (1) to (2) is a warm start since it requires processing fewer nodes to find a solution than Branch and Bound alone.

Generalizing

Consider the following two MILP's:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t. } & x \in P^0 \cap \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ \end{aligned} \qquad \text{(MILP)} \qquad \begin{aligned} \min \quad & \bar{c}^T x \\ \text{s.t. } & x \in \bar{P}^0 \cap \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ \end{aligned} \qquad \text{($\overline{\text{MILP}}$)}$$

where

$$P^0 = \{x \in \mathbb{R}^n : Ax \geq b\} \qquad \bar{P}^0 = \{x \in \mathbb{R}^n : Ax \geq \bar{b}\}$$

We'll make the following assumptions about (MILP) and ($\overline{\text{MILP}}$):

- Either $c = \bar{c}$ or $b = \bar{b}$.
- (MILP) has been solved with Branch and Bound.
- ($\overline{\text{MILP}}$) is unsolved.

We will warm start the solve for ($\overline{\text{MILP}}$) by applying the disjunction from (MILP) to it.

Describing Disjunction

- Let B be the indices of nodes in (MILP)'s Branch and Bound tree (1 indexes the root node).
- Define the feasible region of the LP relaxation for node $t \in B$ as

$$P^t := \{x \in P : u^t \geq x \geq \ell^t\}$$

  where u^t and ℓ^t are bounds on x from branching.
- Applying the disjunction from (MILP) to ($\overline{\text{MILP}}$) gives ($\overline{\text{MILP}}$) a Branch and Bound tree with the following LP relaxation feasible region in node t:

$$\bar{P}^t = \{x \in \bar{P} : u^t \geq x \geq \ell^t\}$$

- Since they share the same disjunction, (MILP) and ($\overline{\text{MILP}}$) have Branch and Bound trees that share the same set of indices, B.
- Let $\mathcal{T}_1 \subseteq B$ represent the set of indices of terminal nodes (i.e. the leaves of the subtree rooted at node 1).

When Things Aren't So Simple

- In practice, applying (MILP)'s disjunction to ($\overline{\text{MILP}}$) rarely yields an immediate optimal solution.
- To find one, we can
  - Place ($\overline{\text{MILP}}$)'s nodes with indices in $\mathcal{T}_1$ into the Branch and Bound queue.
  - Solve ($\overline{\text{MILP}}$) with Branch and Bound with the preloaded queue.
- When $|\mathcal{T}_1|$ is large or there are large differences between c and $\bar{c}$ or between b and $\bar{b}$
  - Many nodes with indices in $\mathcal{T}_1$ may not help find the solution.
  - It could be faster to solve ($\overline{\text{MILP}}$) without applying (MILP)'s disjunction.

We desire a way to refine the search space for Branch and Bound while solving ($\overline{\text{MILP}}$) without having to process the potentially many nodes associated with $\mathcal{T}_1$.

Striking a Balance

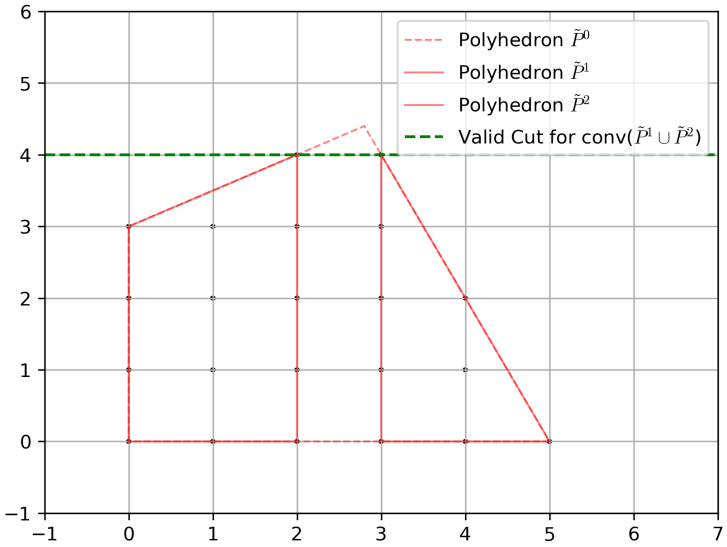


Figure 4: A cut valid for both of (2)'s subproblems

- We can achieve this goal by
  - Tightening the root relaxation, $\bar{P}^1$, with cuts valid for all $\bar{P}^t$ with $t \in \mathcal{T}_1$.
  - Starting Branch and Bound with only the tightened $\bar{P}^1$ in the queue.
- The above cuts are known as **Disjunctive cuts**.
- To find disjunctive cuts, we need
  - A generator to create a disjunctive cut.
  - A cut generation algorithm to run the generator.

We can create disjunctive cuts by implementing the Cut Generating LP (CGLP) and embedding it within a standard cut generation algorithm.

Deriving Disjunctive Cuts

- Let $\mu^t \in \mathbb{R}^m_+$, $w^t \in \mathbb{R}^n_+$, $v^t \in \mathbb{R}^n_+$, and I_n be the n-dimensional identity matrix.
- The following is valid for all $x \in \bar{P}^t$:

$$(A^T\mu^t + I_n w^t - I_n v^t)x \geq {\mu^t}^T \bar{b}^t + {w^t}^T \ell^t - {v^t}^T u^t \tag{3}$$

- Let $\pi \in \mathbb{R}^n$ and $\pi_0 \in \mathbb{R}$ satisfy the following:

$$\begin{aligned} \pi &\geq A^T\mu^t + I_n w^t - I_n v^t \\ \pi_0 &\leq {\mu^t}^T \bar{b}^t + {w^t}^T \ell^t - {v^t}^T u^t \end{aligned} \quad \text{for all } t \in \mathcal{T}_1 \tag{4}$$

- Since (3) is valid for all $x \in \bar{P}^t$ and $t \in \mathcal{T}_1$, it follows (π, π_0) is a valid inequality for all $\bar{P}^t$ with $t \in \mathcal{T}_1$.

We conclude (π, π_0) is a disjunctive cut for $(\overline{\text{MILP}})$.

Deriving the Cut Generating LP

- We seek disjunctive cuts that tighten $\bar{P}^1$ as much as possible.
- Let $x^* = \text{argmin}\{\bar{c}^T x : x \in \bar{P}^1\}$.
- We define the CGLP given the set of nodes with indices in $\mathcal{T}_1$ as follows:

$$
\begin{aligned}
\text{maximize} \quad & \pi_0 - \pi^T x^* \\
\text{subject to} \quad & \pi \geq A^T \mu^t + I_n w^t - I_n v^t && t \in \mathcal{T}_1 \\
& \pi_0 \leq {\mu^t}^T \bar{b}^t + {w^t}^T \ell^t - {v^t}^T u^t && t \in \mathcal{T}_1 \\
& 1 = \sum_{t \in \mathcal{T}_0} \Big(\sum_{j=1}^{m} \mu_j^t + \sum_{i=1}^{n} w_i^t + \sum_{i=1}^{n} v_i^t \Big) \\
& \mu^t \in \mathbb{R}^m_+,\ w^t \in \mathbb{R}^n_+\ v^t \in \mathbb{R}^n_+ && t \in \mathcal{T}_1
\end{aligned}
\qquad \text{(CGLP-}\mathcal{T}_1\text{)}
$$

We can solve CGLP-$\mathcal{T}_1$ repeatedly within a cutting plane algorithm to generate a set of increasingly tighter disjunctive cuts.

A Cutting Plane Algorithm Using the CGLP

We can define a cutting plane algorithm to tighten $\bar{P}^1$ with disjunctive cuts valid for all $\bar{P}^t$ with $t \in \mathcal{T}_1$ as follows:

Algorithm CPA-$\bar{P}^1$-$\mathcal{T}_1$

$\bar{P}^{1,0} := \bar{P}^1$
$k := 0$
while True **do**:
    $x^* = \text{argmin}\{\bar{c}^T x : x \in \bar{P}^{1,k}\}$
    Let (π^k, π_0^k) be the optimal solution to (CGLP-$\mathcal{T}_1$)
    if $\pi_0^k - {\pi^k}^T x^* \leq 0$ **then**:
        stop
    end if
    $\bar{P}^{1,k+1} = \bar{P}^{1,k} \cap \{x \in \mathbb{R}^n : {\pi^k}^T x \geq \pi_0^k\}$
    $k = k + 1$
end while
$\bar{P}^1 = \bar{P}^{1,k}$

We can warm start Branch and Bound for ($\overline{\text{MILP}}$) by using the updated $\bar{P}^1$ as the root relaxation.

Putting It All Together

We did the following to warm start solving ($\overline{\text{MILP}}$) from the solve of (MILP):

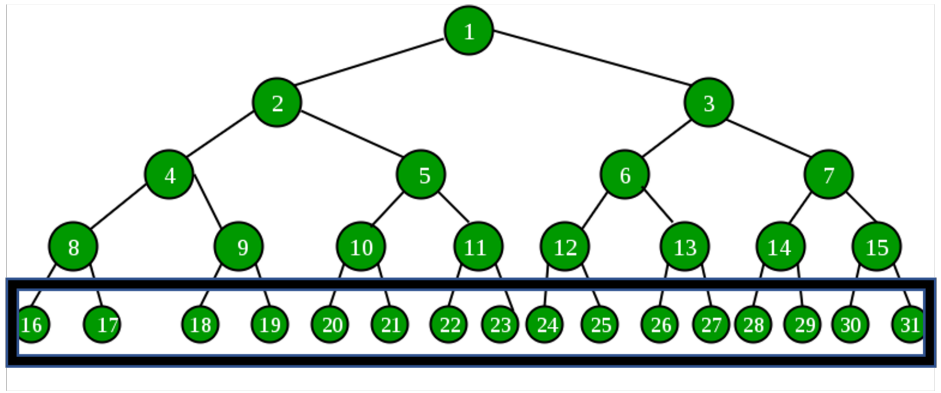
- Apply the disjunction from (MILP) to ($\overline{\text{MILP}}$), yielding
  - A LP relaxation feasible region $\bar{P}^t$ for all $t \in B$
  - A set of terminal nodes with indices $\mathcal{T}_1$
- Formulate (CGLP-$\mathcal{T}_1$) to find a cut valid for all $\bar{P}^t$ with $t \in \mathcal{T}_1$.
- Run **CPA**-$\bar{P}^1$-$\mathcal{T}_1$ to tighten $\bar{P}^1$.
- Solve ($\overline{\text{MILP}}$) with Branch and Bound starting from the tightened $\bar{P}^1$.

For this approach to be effective, **CPA**-$\bar{P}^1$-$\mathcal{T}_1$ needs to run in less time than that saved from warm starting ($\overline{\text{MILP}}$)'s solve.

Computational Considerations

Handling Disjunctions with Many Terms

- Consider the following branch and bound tree with terminal nodes $\mathcal{T} = \{16, ..., 31\}$.
- CGLP-$\mathcal{T}$ has $|\mathcal{T}|m$ variables and $|\mathcal{T}|(2n + m)$ constraints.
- Solving CGLP-$\mathcal{T}$ can become intractible for MILPs where m, n or $|\mathcal{T}|$ is large.



We need a way to decompose the CGLP into solvable subproblems that still yield cuts valid for the root node.

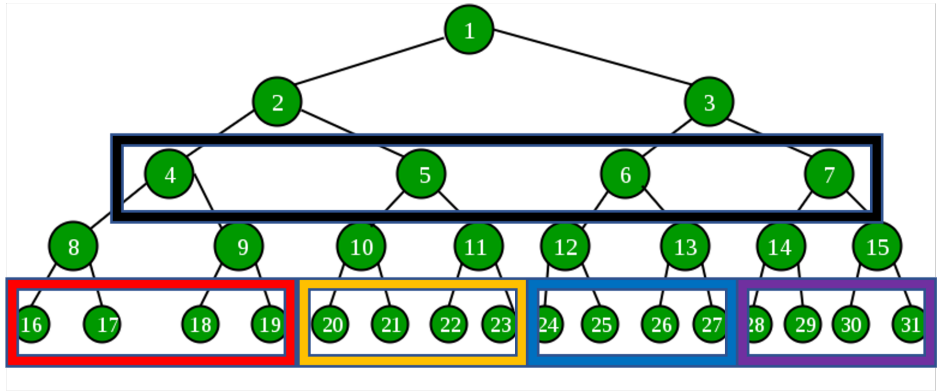
Handling Disjunctions with Many Terms

Instead of running **CPA**-$\bar{P}^1$-$\mathcal{T}_1$, we can

- Run **CPA**-$\bar{P}^4$-$\mathcal{T}_4^2$, **CPA**-$\bar{P}^5$-$\mathcal{T}_5^2$, **CPA**-$\bar{P}^6$-$\mathcal{T}_6^2$, and **CPA**-$\bar{P}^7$-$\mathcal{T}_7^2$.
- Then run **CPA**-$\bar{P}^1$-$\mathcal{T}_1^2$.

The result of this is that

- More CGLPs will be solved.
- Each solved CGLP is significantly smaller.



Let $\mathcal{T}_1^2 = \{4, ..., 7\}$, $\mathcal{T}_4^2 = \{16, ..., 19\}$, $\mathcal{T}_5^2 = \{20, ..., 23\}$, $\mathcal{T}_6^2 = \{24, ..., 27\}$, and $\mathcal{T}_7^2 = \{28, ..., 31\}$.

We're exploring trade-offs between having fewer CGLP's to solve and solving smaller CGLP's to find the most effective way to refine the root node relaxation.

Open Source Contributions

Implementation Considerations

Goals

- We need the following attributes to implement the CGLP:
  - The branch and bound tree
  - Each node's LP relaxation
- We would like a Python implementation because Python
  - Is quick to write
  - Has a large community

Limitations

- No solver has a Python API with these attributes.
- Unsure of solver persisting these attributes.

Solutions

- Add these attributes to CBC/CyLP.
- Both are open source.
- CBC has the attributes, but they need persistence.
- CyLP converts CBC into Python.

We can add to CyLP and persist in CBC the attributes we need for the CGLP to make them available in Python.

CyLP Motivations

Because CyLP is a Cython package, its classes extend various classes of COIN-OR

- CyClpSimplex extends ClpSimplex.
- CyCbcModel extends CbcModel.

How Cython works:

- Python objects instantiate by binding a corresponding C/C++ instance.
- Function calls can call functions on the bound C/C++ instance.
- Attribute access can access attributes on the bound C/C++ instance.
- Any C/C++ code used by Python objects is compiled during packaging.

Advantages of using Cython:

- It makes existing C/C++ classes available in Python.
- It combines fast development of Python with fast execution of C/C++

Introduction to Cython

Below is a typical example of how instantiation works in Cython:

```
75      def __cinit__(self, cyLPModel=None):
76          self.CppSelf = new CppIClpSimplex(<cpy_ref.PyObject*>self,
77                                  <runIsPivotAcceptable_t>RunIsPivotAcceptable,
78                                  <varSelCriteria_t>RunVarSelCriteria)
79          self.vars = []
80          self.coinModel = CyCoinModel()
81
82          self.cyLPModel = cyLPModel
83          if cyLPModel:
84              if isinstance(cyLPModel, CyLPModel):
85                  self.loadFromCyLPModel(cyLPModel)
86              else:
87                  raise TypeError('Expected a CyLPModel as an argument to ' \
88                                  'cylpSimplex constructor. Got %s' %
89                                  cyLPModel.__class__)
```

Figure 5: CyClpSimplex objects bind an instance of CppIClpSimplex to the `.cppSelf` attribute during initialization.

```
616 IClpSimplex::IClpSimplex(PyObject *obj_arg, runIsPivotAcceptable_t runIsPivotAcceptable_arg,
617                          varSelCriteria_t runVarSelCriteria ):ClpSimplex()
618                         {
619
620     _import_array();
621     tempArrayExists = false;
622     obj = obj_arg;
623     runIsPivotAcceptable = runIsPivotAcceptable_arg;
624     varSelCriteria = runVarSelCriteria;
625     customPrimal = 0;
626     createStatus();
627     pinfo = ClpPresolve();
628
629     tempRow = NULL;
630     tempRow_vector = NULL;
631     QP_BanList = NULL;
632     QP_ComplementarityList = NULL;
633 }
```

Figure 6: CppIClpSimplex is an alias for IClpSimplex, a subclass of ClpSimplex.

Introduction to Cython

Below is a typical example of how function calls work in Cython:

```
1484    cpdef int readMps(self, filename, int keepNames=False,
1485            int ignoreErrors=False) except *:
1486        '''
1487        Read an mps file. See this :ref:`modeling example <modeling-usage>`.
1488        '''
1489        filename = filename.encode('ascii')
1490        name, ext = os.path.splitext(filename)
1491        if ext not in [b'.mps', b'.qps']:
1492            print('unrecognised extension %s' % ext)
1493            return -1
1494
1495        if ext == b'.mps':
1496            return self.CppSelf.readMps(filename, keepNames, ignoreErrors)
1497        else:
1498            m = CyCoinMpsIO.CyCoinMpsIO()
1499            ret = m.readMps(filename)
1500            self._Hessian = m.Hessian
1501
1502            # Since CyCoinMpsIO.readMps seems to be different from ClpModle.readMps
1503            # for the moment we read the problem again
1504            # FIXME: should be fixed
1505            #self.loadProblem(m.matrixByCol, m.variableLower, m.variableUpper,
1506            #                 m.objCoefficients,
1507            #                 m.constraintLower, m.constraintUpper)
1508            #return ret
1509
1510            return self.CppSelf.readMps(filename, keepNames, ignoreErrors)
```

Figure 7: Reading a `.mps` file in CyLP

- CyClpSimplex reads `.mps` files by calling on ClpSimplex's eponymous function.
- Solving LP's with CyClpSimplex works analogously.

Introduction to Cython

Below is a typical example of how attribute access works in Cython:

```
295     property primalVariableSolution:
296         '''
297         Solution to the primal variables.
298
299         :rtype: Numpy array
300         '''
301         def __get__(self):
302             ret = <object>self.CppSelf.getPrimalColumnSolution()
303             if self.cyLPModel:
304                 m = self.cyLPModel
305                 inds = m.inds
306                 d = {}
307                 for v in inds.varIndex.keys():
308                     d[v] = ret[inds.varIndex[v]]
309                     var = m.getVarByName(v)
310                     if var.dims:
311                         d[v] = CyLPSolution()
312                         dimRanges = [range(i) for i in var.dims]
313                         for element in product(*dimRanges):
314 #                           d[v][element] = ret[var.__getitem__(element).indices[0]]
315                             d[v][element] = ret[var.fromInd+var[element].indices[0]]
316                 ret = d
317             else:
318                 names = self.variableNames
319                 if names:
320                     d = CyLPSolution()
321                     for i in range(len(names)):
322                         d[names[i]] = ret[i]
323                     ret = d
324             return ret
```

- CyClpSimplex's PrimalVariableSolution attribute returns the corresponding ClpSimplex attribute.
- Cython functions handle converting C/C++ types into Python types (e.g. `<object>` in line 302)

Figure 8: Accessing an LP primal solution in CyLP

Closing Thoughts

Putting It All Together

To recap, the CGLP generates cuts that retain primal and dual bounds from a MILP solve while resetting the underlying disjunction to a single node.

What we've accomplished:

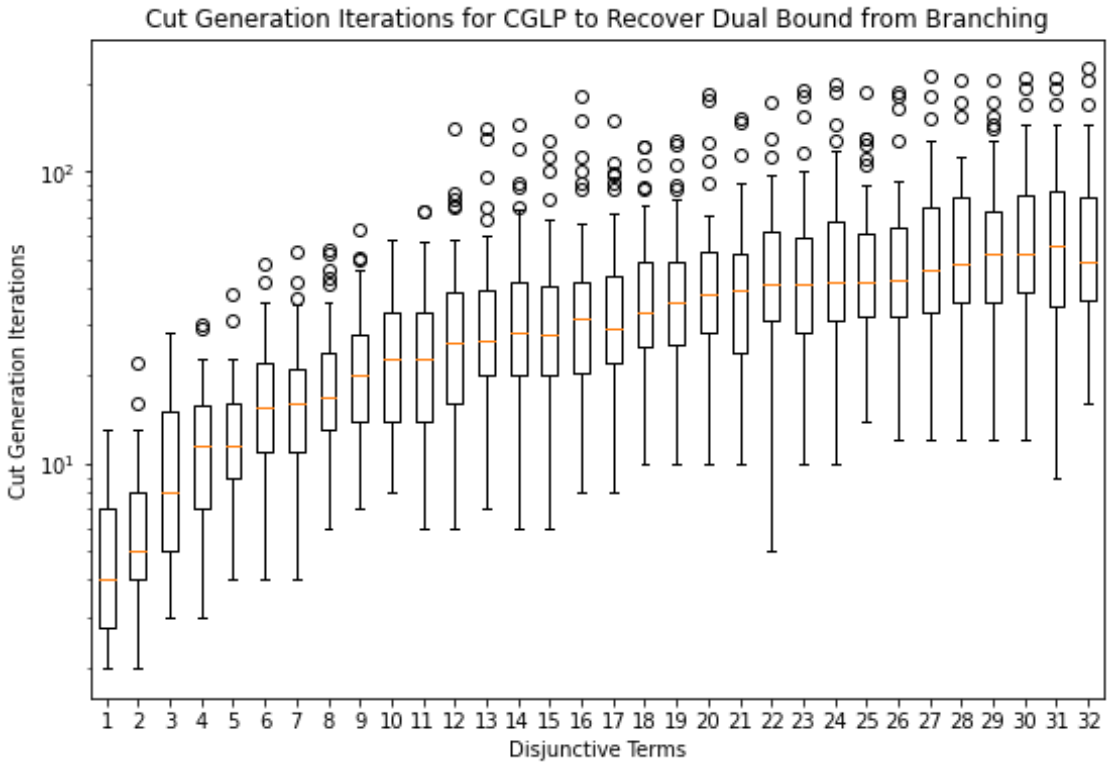
- Implemented CGLP class in Python
- Developed algorithm to decompose CGLP for large disjunctions
- Added necessary branch and bound tree attributes to CyLP and CBC

What's next:

- Integrate each accomplishment
- Test warm start effectiveness for MILP's with similar objectives or constraint bounds
- Further tune CGLP performance
- Test warm start effectiveness for MILP's with added variables or constraints

Index

Numerical Experiments

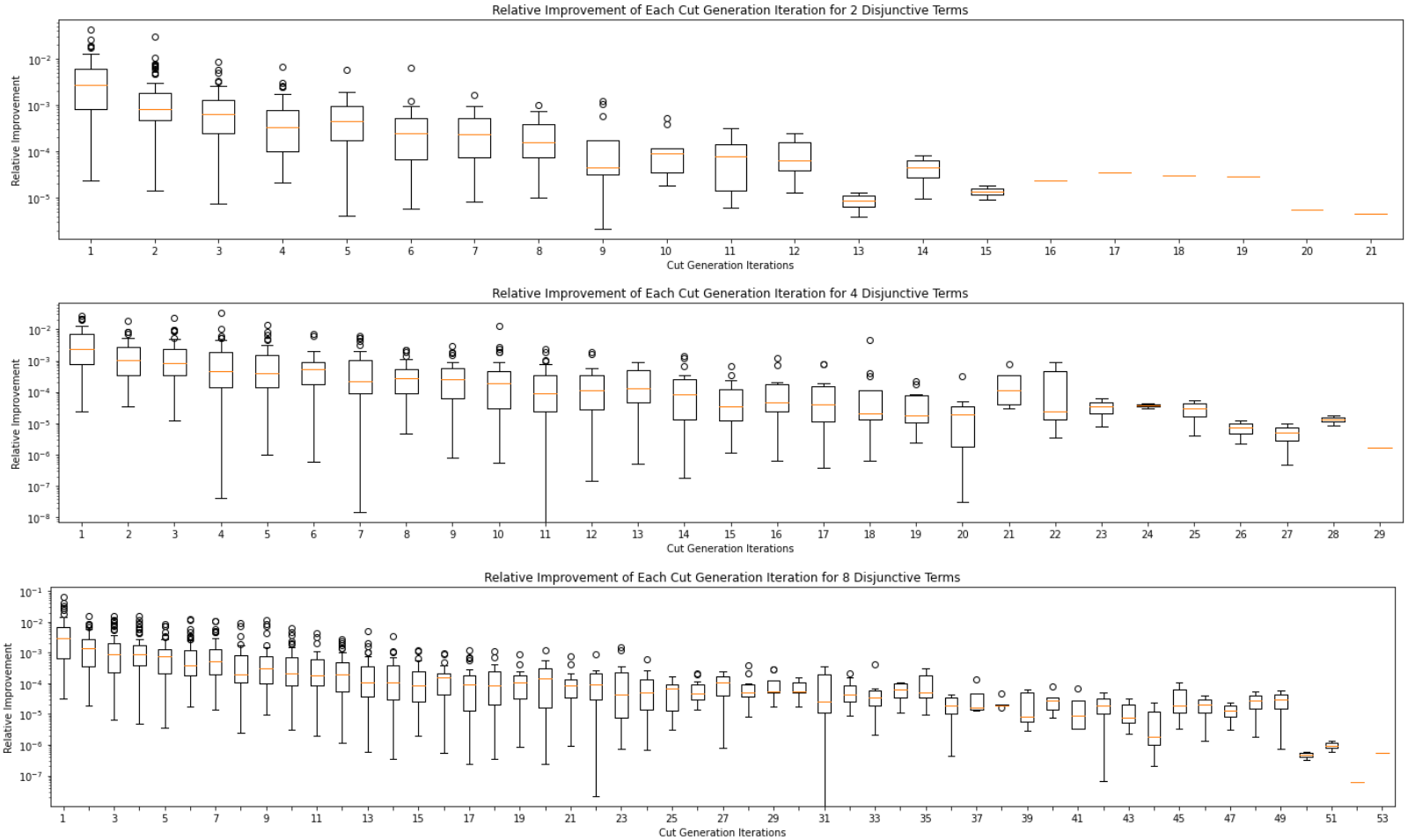


- We also need to consider how many iterations it takes a cutting plane algorithm to yield tight cuts.
- We see a linear relationship between disjunctive terms and number of cut generation iterations for each CGLP to recover the disjunction's dual bound.

Knowing the convergence rate to a disjunction's dual bound for a cut generation algorithm using the CGLP may allow us to infer which size disjunctions to use.

Numerical Experiments

Cut generation algorithms using the CGLP to recover the dual bound for the given sized disjunctions below converge at similar rates.



No immediate advantages appear among different sized disjunctions and the number of cutting plane algorithm iterations required to derive tight cuts.

Restarting MILP's

- Warm starting with a tightened root LP relaxation can be preferred to a partial tree already found on a looser root relaxation.
- We aim to identify problem classes where there is a strong preference for the former.
- When this is true, a partial solve followed by warm start method (c) takes less time than solving the instance cold.
- The idea then is as follows:
  - Solve LP relaxations for certain number of nodes.
  - Generate CGLP cuts against this tree.
  - Tighten root node with these cuts.
  - Restart Branch and Cut from beginning with tightened root relaxation.

If many problem classes exist where the above is beneficial, this would dramatically change how solvers work today.

Branch and Price

- We can tighten the LP relaxation for a MILP with the following formulation:

$$\max_{x \in \text{conv}(\mathcal{S}_R)} \{c^T x | A'' x \leq b''\} \tag{5}$$

where

$$\mathcal{S}_R = \{x \in \mathbb{Z}^n | A' x \leq b'\}$$

- With $\mathcal{E}$ as the set of extreme points of $\text{conv}(\mathcal{S}_R)$, we can reformulate as follows:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & \sum_{s \in \mathcal{E}} \lambda_s s = x \\ & A'' x \leq b'' \\ & \sum_{s \in \mathcal{E}} \lambda_s = 1 \\ & \lambda \in \mathbb{R}_+ \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \max \quad & \sum_{s \in \mathcal{E}} \lambda_s (c^T s) \\ \text{s.t.} \quad & \sum_{s \in \mathcal{E}} \lambda_s (A'' s) \leq b'' \\ & \sum_{s \in \mathcal{E}} \lambda_s = 1 \\ & \lambda \in \mathbb{R}_+ \end{aligned} \tag{DWLP}$$

Branch and Price

- We solve (DWLP) by column generation, only ever using a subet of $\mathcal{E}$. For dual values u and α of (DWLP)'s constraints, we find a new s to augment the current subset by solving the following:

$$-\alpha + \max_{x \in \mathcal{S}_R} \{(c - uA'')x\} \qquad (\text{LR}(u))$$

- (DWLP) is often the LP relaxation in Branch and Price. Notice (LR(u)) represents a series of closely related MILP's.
  - Within a single solve of (DWLP), each (LR(u)) differs only by objective.
  - A common branching technique in Branch and Price is to bound variables in (LR(u)). Thus, for fixed u, (LR(u)) differs from its ancestor nodes by its RHS.

By warm starting the pricing problem in Branch and Price, we may be able to expand the sets of problems solved by this algorithm.

Warm Starting Big Picture

We have a couple approaches at our disposal for warm starting a MILP from a previously solved one.

(a) Good warm start:

- Recalculate the primal bound.
- Restart Branch and Cut from the root node.

(b) Better warm start:

- Parameterize cuts.
- Recalculate the primal and/or dual bounds.
- Restart Branch and Cut from the leaf nodes.

(c) Hypothesized best warm start:

- Parameterize cuts.
- Add to the root relaxation cuts valid for all terminal subproblems.
- Restart Branch and Cut from the root node.

Next Steps

There are a few other problem classes we look to test the effectiveness of warm starts:

- Multiobjective MILP
- Primal Heuristics (RINS)
- Bilevel MILP
- Real-Time MILP
- MINLP

We have the following next tasks in mind:

- Make cut generation numerically safe.
- Parametrize cut generation.
- Implement algorithms for each problem class.
- Run experiments for each algorithm comparing warm started version to current algorithms.

We intend to test our warm starting methods against the problem classes mentioned in this presention to determine which can leverage these approaches to more effeciently be solved.

Simple Approach to Warm Starting

- Warm starting means to reuse a Branch and Bound tree from solving a previous MILP when solving a new one.
- One approach is to reevaluate leaf nodes' solutions at the new MILP's objective and RHS to set applicable bounds. Then leaf nodes are placed into a priority queue, and Branch and Cut resumes.
- This approach has the following advantages:
  (a) When only the objective changes, all previously feasible solutions remain feasible. We can start the primal bound as the one with the best evaluation of the new objective.
  (b) When only the RHS changes, each node's dual LP relaxation maintains the same feasible region. We can start the dual bound of each node as its previous dual LP solution evaluated at the new RHS.
  (c) Restarting from the leaf nodes prevents us from having to branch and bound to regenerate them in the new instance.

Simple Approach to Warm Starting

- Warm starting means to reuse a Branch and Bound tree from solving a previous MILP when solving a new one.
- One approach is to reevaluate leaf nodes' solutions at the new MILP's objective and RHS to set applicable bounds. Then leaf nodes are placed into a priority queue, and Branch and Cut resumes.
- This approach has the following disadvantages:
  (a) The primal or dual bound is weak if the objective or RHS changes significantly.
  (b) Potentially many leaf nodes will need to be branched.
  (c) Generated cuts may no longer be valid if RHS changed.

Starting a MILP solve with a Branch and Bound tree from a previous solve can be a simple and quick way to improve performance, but such an approach is not a panacea.

Motivating the Cut Generating LP

- Generating strong cuts of the convex hull of the previous disjunctive terms' LP relaxations near the new optimal solution ameliorates disadvantages (a) and (b). Instead of reusing the previous tree to solve our problem, we solve with a new Branch and Bound tree with these cuts added to the root LP relaxation.
- This removes advantages (b) and (c) but tightens the root LP relaxation. In many cases this is preferred because it can more quickly give the solver an idea of where to search for an optimal solution.
- These cuts are the valid inequalities that violate by as much as possible our best estimates of the new optimal solution.
- Given such estimates, the Cut Generating LP (CGLP) finds such inequalities.

It may be possible to improve a warm start by replacing the previous tree with strategically chosen strong cuts of the convex hull of its disjunctive terms' LP relaxations.

Handling Constraint Bound Changes

- Should $b^k \neq b^{k'}$, (A^{kt}, b^{kt}) may no longer be valid for the matching disjunctive term in MILP k'. Thus, MILP k's tree may not be able to warm start MILP k'.
- Let $p_{k't}(A^{kt}, b^{kt}) = (p_{k't}(A, b^k), p_{k't}(\Pi^{kt}, \Pi_0^{kt})) : \mathbb{R}^{m^t \times n+1} \rightarrow \mathbb{R}^{m^t \times n+1}$ map the constraints (A^{kt}, b^{kt}) to constraints that are valid for matching disjunctive term in MILP k', where
  - $p_{k't}(A, b^k) = (A, b^{k'})$
  - $p_{k't}(\Pi^{kt}, \Pi_0^{kt})$ updates each GMI cut as described in Guzelsoy (2006) and each CGLP cut (π, π_0) as follows. Let the tree of MILP κ be from which (π, π_0) is derived. Let $(\tilde{A}, \tilde{b}) = p_{k't}(A^{\kappa t}, b^{\kappa t})$. Then

$$\begin{aligned}(\pi, \pi_0) =& (\max\{\tilde{A}^T \mu^{kt} + {I_n}^T w^{kt} - {I_n}^T v^{kt} \text{ for all } t \in \mathcal{T}^k\}, \\ & \min\{\tilde{b}^T \mu^{kt} + {I^{kt}}^T w^{kt} - {u^{kt}}^T v^{kt} \text{ for all } t \in \mathcal{T}^k\})\end{aligned}$$

If $b^k \neq b^{k'}$, cuts generated for MILP k must be made valid for MILP k' before the tree of MILP k can be used to warm start MILP k'.

Warm Starting Big Picture

- There are a couple of different approaches we can take to warm starting a MILP k' from one previously solved, MILP k.
- All look to accomplish the same thing, but they differ from each other in the following ways:
  - When solving MILP k' with the tree from MILP k's solve and only the objectives differ between the two instances, there is no need to transform cuts, and only the feasible leaves need searched. However, the dual function is lost, so search priority is less clear.
  - When solving MILP k' with the tree from MILP k's solve and only the RHS's differ between the two instances, cuts may need to be transformed to remain valid, and all leaves need searched. However, the dual function remains intact, so search priority is clearer.
  - When generating CGLP cuts from MILP k to warm start MILP k', a partially searched tree is exchanged for a tighter root relaxation. Some nodes may be re-evaluated, but the number of nodes to search may be reduced.

Dual Decomposition for Stochastic MILP

- Let $S^j := \{(x, y^j) : Ax \leq b, x \in X, T^j x + wy^j \leq h^j, y^j \in Y\}$
- A multistage stochastic MILP can be formulated as follows:

$$\max \left\{ c^T x + \sum_{j=1}^{r} p^j q^{j\,T} y^j : (x, y^j) \in S^j \text{ for } j \in [r] \right\} \tag{6}$$

- It can be reformulated as follows:

$$\max \left\{ \sum_{j=1}^{r} p^j (c^T x^j + q^{j\,T} y^j) : (x^j, y^j) \in S^j \text{ for } j \in [r], x^1 = \ldots = x^r \right\} \tag{7}$$

- For H^j such that $\sum_{j=1}^{r} H^j x^j = 0$ is equivalent to $x_1 = \ldots = x_r$, (6) and (7) can have an upper bound decomposed as follows:

$$\min_{\lambda} D(\lambda) \tag{LD}$$

$$D(\lambda) = \max \left\{ \sum_{j=1}^{r} p^j (c^T x^j + q^{j\,T} y^j) + \lambda (H^j x^j) : (x^j, y^j) \in S^j \text{ for } j \in [r] \right\}$$

Dual Decomposition for Stochastic MILP

- $D(\lambda)$ consists of r independent MILP's.
- (LD) is solved by a subgradient algorithm. We branch on the average value of x in (LD)'s solution and bound (LD) again until x^j are all identical.
- Notice there are r series of MILPs sharing similar structure:
  - MILP j in $D(\lambda)$ differs from MILP j in $D(\lambda')$ only by objective.
  - MILP j in $D(\lambda)$ before branching differs from MILP j in $D(\lambda)$ after branching only by variable bounds.

Since solving a multistage Stochastic MILP involves solving multiple series of potentially difficult MILPs, the warm starting techniques presented earlier could be beneficial to the solution method.